Computing the Chromatic Polynomials of the Six Signed Petersen Graphs

July 29, 2012

Erika Meza
Loyola Marymount University
Bryan Nevarez
Queens College CUNY
Alana Shine
Pomona College

Mathematical Sciences Research Institute
Undergraduate Program
Summer 2012

Abstract

Graphs are a collection of vertices and edges that connect some vertices to others. Signed graphs are graphs whose edges are assigned positive or negative labels and may contain loops. Signed graphs have been useful in understanding phenomena that occur in our society, such as interactions within a group of individuals or the representation of biological networks which allow for further analysis of the relationships that exist in nature. Our work addresses open questions regarding proper colorings of signed graphs, in which the vertices are assigned colors based on rules according to the edge connections and edge labels. We explore the number of proper colorings of these graphs by computing their corresponding chromatic polynomials. In particular, we investigate the six distinct signed Petersen graphs studied by Thomas Zaslavsky (Discrete Math 312 (2012), no 9, 1558-1583) and prove his conjecture that they have distinct chormatic polynomials.

1 Introduction

The Petersen graph, depicted in Figure 1, is comprised of 10 vertices and 15 edges. It serves as a reliable reference point to many proposed theorems in graph theory. Considering signed versions of the Petersen graph, Thomas Zaslavsky

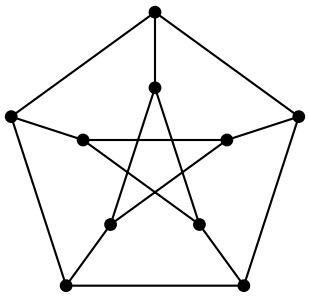


Figure 1: The Petersen Graph.

ascertains that while there are 2^{15} ways to assign signs to the fifteen edges, only six of these are different up to switching isomorphism [3]. (We will give precise definitions of signed graphs and switching in Section 2).

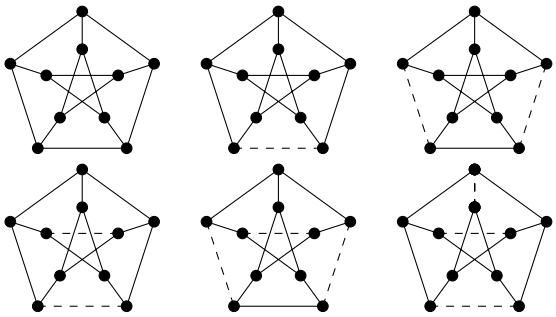


Figure 2: Zaslavsky's six signed Petersen graphs

Zaslavsky conjectures that the chromatic polynomials of the six signed Petersen graphs shown in Figure 2 will differ from each other. We explore ways to verify this conjecture through computations of all six chromatic polynomials. Our objective is to develop and execute a computer program that will efficiently determine the number of proper k-colorings for these six signed Petersen graphs. We explore some basic properties of smaller signed graphs to help develop an efficient algorithm to compute the six signed Petersen graphs.

2 Signed Graph Terminology

The terms and definitions used throughout this paper are adopted from those in [2].

A *signed graph* $\Sigma := (V, E, \sigma)$ consists of vertices, denoted by V, edges, denoted by E, and a function $\sigma : E \to \{+, -\}$ which is a signature that labels each edge either positive or negative. We will use $\sigma(ij) = +$ or $-$ for edge $ij \in E$

throughout the paper. Σ may contain *negative loops* which are negative edges that connect a vertex to itself. Σ also may contain *multiple edges* which are edges that are incident to the same two vertices if the two edges have opposite sign.

Let $\Sigma = (V, E, \sigma)$ and $v \in V$. A *switching function* s is a mapping from (Σ, v) to $\Sigma' = (V, E, \sigma')$ such that $\sigma'(ij) = \sigma(ij)$ if ij is not incident to v and $\sigma'(ij) = -\sigma(ij)$ if ij is incident to v. That is, switching Σ at v negates the sign of all edges incident to v.

The signed graphs $\Sigma_1 = (V_1, E_1, \sigma_1)$ and $\Sigma_2 = (V_2, E_2, \sigma_2)$ are *isomorphic* if there is a graph isomorphism $\psi : V_1 \to V_2$ such that $\sigma_1(ij) = \sigma_2(\psi(i)\psi(j))$. Two signed graphs Σ_1 and Σ_2 are *switching isomorphic* if there exists a way to switch vertices in Σ_1 to get a signed graph that is isomorphic to Σ_2. Zaslavsky shows that there exists only the six signed Petersen graphs, shown in Figure 2, up to switching isomorphism.[3].

Let Σ be a signed graph. A *proper k-coloring* of Σ is a mapping x: $V \to \{-k, -k+1, -k+2, ..., 0, 1, 2, ..., k\}$ such that $x_i \neq \sigma(ij)x_j$ for any $ij \in E$. That is, if ij is a positive edge then $x_i \neq \sigma(ij)x_j = x_j$, and if ij is a negative edge then $x_i \neq \sigma(ij)x_j = -x_j$. Thus, adjacent vertices on any positive edge cannot be colored the same color and adjacent vertices on any negative edge cannot be colored opposite colors.

The *chromatic polynomial* $c_\Sigma(k)$ is a function that when evaluated at any positive integer k gives the number of proper k-colorings of Σ. (We will recall a deletion-contraction theorem in Section 3.2, from which it follows that the $c_\Sigma(k)$ is indeed a polynomial in k.) Zaslavsky observed that the chromatic polynomial of Σ is preserved under switching, and thus switching-isomorphic graphs have the same chromatic polynomial [3].

Theorem 1. *If Σ and Σ' are switching isomorphic, then $c_\Sigma(k) = c_{\Sigma'}(k)$.*

Proof. It suffices to show that the theorem holds when Σ is switched at a single vertex. Let $\Sigma = (V, E, \sigma)$ be a signed graph and $i \in V$. Let $\Sigma' = (V, E, \sigma')$ be the signed graph Σ with vertex i switched. Let A be the set of proper k-colorings of Σ, and let B be the set of proper k-colorings of Σ'. Let $f : A \to B$ be defined as $f([x_1, ..., x_i, ..., x_n]) = [x_1, ..., -x_i, ..., x_n]$. Given a proper coloring, $[x_1, ..., x_i, ..., x_n]$, of Σ it follows that $x_i \neq \sigma(ij)x_j$. Upon switching Σ at vertex i note that the vertices on edges that are not incident to i in Σ remain the same proper colors in Σ' and $\sigma'(ij) = -\sigma(ij)$. So $-x_i \neq -\sigma(ij)x_j = \sigma'(ij)x_j$. Hence, $[x_1, ..., -x_i, ..., x_n] \in B$. Since the negative of an integer is unique, it follows that f assigns an element in B to a unique element in A. A similar argument applies when Σ' is switched at a vertex. Therefore, f is a bijection between the proper colorings in Σ and in Σ'. □

3 SAGE Deletion and Contraction Program

Computing the chromatic polynomials of the six signed Petersen graphs incorporates the Deletion-Contraction Theorem (Theorem 2 below) into a Python-based mathematical software called SAGE [1]. The following sections illustrate the individual components of the program.

3.1 Incidence Matrix

The first step in our program is finding an alternate way to represent the signed graphs other than their commonly known visual representation. Incidence matrices allow one to compactly store the description of a graph. To create an incidence matrix of Σ, rewrite Σ as a bidirected graph G. The bidirections assigned to the edges of G correspond to the signs of the edges in Σ as shown in Figure 3. A positive edge ab has two arrows pointing towards the same direction and a negative edge ac has two arrows pointing in opposite directions.

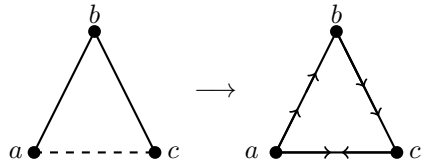


Figure 3: Signed graph Σ turned into an bidirected graph.

This convention provides a method to write the entries of Σ's incidence matrix. The rows of an incidence matrix represent vertices of Σ and the columns represent edges of Σ as shown in Figure 4. A vertex on an edge whose closest arrow is pointing towards it receives a 1 in its (i, j) entry where i is the vertex and j is the edge incident i is on. A -1 is placed in the (i, j) entry of the incidence matrix when the closest arrow to a vertex on the edge is pointing away from the vertex. These are the only non-zero entries of an incidence matrix. As a result, columns that represent a positive edge have a single -1 and a single 1 as the only nonzero entries. On the other hand, columns representing negative edges only have two nonzero entries that are either both 1's or both -1's. Incidence matrices can be easily manipulated by a program making them useful for computing chromatic polynomials.

	ab	bc	ac
a	-1	0	-1
b	1	-1	0
c	0	1	-1

Figure 4: Incidence Matrix for the graph in Figure 3.

3.2 Deletion-Contraction

In order to compute the chromatic polynomials for the six signed Petersen graphs, we first break these down into smaller graphs.

Let $\Sigma = (V, E, \sigma)$ be a signed graph and $e = ij \in E$. The *deletion* of an edge e in Σ, denoted by $\Sigma - e$, removes e from Σ to create a new graph. The *contraction* of a positive edge e in Σ, denoted by $\Sigma\backslash e$, collapses edge ij by replacing vertices i and j with a new vertex h. Now any vertices adjacent to i or j are adjacent to h. This is equivalent to setting $i = j$ because now the two vertices act as the same vertex. The contraction on a negative loop at vertex i in Σ creates a new graph with i deleted from V and replaces every edge incident to i with negative loops. The two operations are depicted in Figure 5.

Deletion reduces a graph by removing an edge, while contraction removes both an edge and a vertex. There is a recurrence relation between the chromatic polynomial of Σ and the chromatic polynomials of Σ with e deleted and contracted as follows:

Theorem 2 (Deletion-Contraction Theorem)**.** *Let Σ be a signed graph, and let e be a positive edge or negative loop of Σ.*

$$c_\Sigma(k) = c_{\Sigma - e}(k) - c_{\Sigma\backslash e}(k).$$

Proof. Case 1: Let e be a positive edge ij in Σ. Then the chromatic polynomial of Σ yields the number of proper k-colorings when x_i, the color of vertex i, does not equal x_j, the color of vertex j. The proper coloring of any edge not incident to i and j will not be altered by deletion and contraction. However, removing e from Σ gives us a new graph $\Sigma - e$, such that $c_{\Sigma-e}(k)$ counts the colorings in the case that $x_i = x_j$ and also in the case when $x_i \neq x_j$. Contracting edge ij yields $c_{\Sigma\backslash e}(k)$ which only counts the colorings when $x_i = x_j$ because i and j have been collapsed into the same vertex. Thus, we know that the proper colorings for $c_{\Sigma-e}(k)$ will be included in either $c_\Sigma(k)$ or in $c_{\Sigma\backslash e}(k)$ such that $c_{\Sigma-e}(k) = c_\Sigma(k) + c_{\Sigma\backslash e}(k)$. Therefore, all the k-colorings we want to exclude from $c_{\Sigma-e}(k)$ to obtain $c_\Sigma(k)$ are counted in $c_{\Sigma\backslash e}(k)$ and $c_{\Sigma-e}(k) - c_{\Sigma\backslash e}(k) = c_\Sigma(k)$.

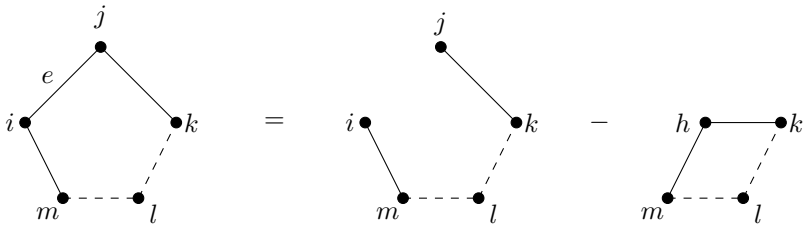


Figure 4: Illustration of $c_\Sigma(k) = c_{\Sigma-e}(k) - c_{\Sigma\backslash e}(k)$ for a positive edge e.

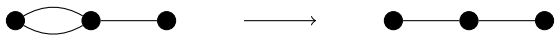
Case 2: e is a negative loop at i. If we can prove $c_{\Sigma-e}(k) = c_\Sigma(k) + c_{\Sigma\backslash e}(k)$ then deletion-contraction holds for a negative loop. $c_{\Sigma-e}(k)$ includes

the colorings where $x_i = 0$ and those where $x_i \neq 0$. $c_\Sigma(k)$ counts the colorings where $x_i \neq 0$ because e at i is a negative loop and x_i can not be colored it's own negative. $c_{\Sigma\setminus e}(k)$ is still under the conditions $x_i \neq \sigma(ij)x_j$. If we set $x_i = 0$, then $x_j \neq 0$. These are the colorings of $c_{\Sigma\setminus e}(k)$ because there is a negative loop at j. So $c_\Sigma(k) + c_{\Sigma\setminus e}(k)$ counts the coloring where $x_i = 0$ and those where $x_i \neq 0$ and is thus equivalent to $c_{\Sigma\setminus e}(k)$. □

The deletion-contraction theorem can be used to compute the chromatic polynomials for the six signed Petersen graphs. However, such computations would require recording the newly created graph after every single deletion and contraction until every graph was simplified down to paths and cycles, of which we know the chromatic polynomials. In other words, if we were to delete and contract every edge we would have 2^{15} graphs to compute the chromatic polynomial of for each of the six signed Petersen graphs. So, to aid in computation, we create a program to perform deletion and contraction on a Petersen graph while recording the deleted and contracted graphs and evaluating the sum of the chromatic polynomials of our simplified graphs.

Before we begin our deletion-contraction process, we want to eliminate any multi-edges that create a redundancy in the set restrictions for a given graph G. For instance, consider the case with vertices i and j and two positive edges e_1 and e_2 connecting them. Then both e_1 and e_2 give us the same information about our vertices, that is, $x_i \neq x_j$. Thus, we want to eliminate one of these edges which are represented as identical columns in our corresponding incidence matrix for G as shown in the example below.

Example 1.



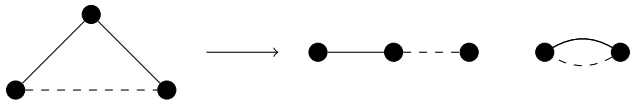
The incidence matrix for a graph with three vertices connecting a multi-edge and another single edge $\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ *is simplified to* $\begin{bmatrix} 1 & 0 \\ -1 & 1 \\ 0 & -1 \end{bmatrix}$ *as shown above.*

Once we have removed all the multi-edges in a graph, the computer algorithm will take the reduced incidence matrix through deletion-contraction depending on which of the following cases the given graph falls under. First, consider the case of a graph Σ consisting of a single edge. Then our incidence matrix will contain a single column with exactly two nonzero entries for the two vertices connected to the edge. In this case, our deletion of the edge will result in a single column with all zero entries and our contraction will yield the same matrix with a single column and one less row. If this does not apply to the given graph Σ, then the program will proceed to next case.

Given that Σ has more than one edge, the program will search for the first positive edge e encountered in the given incidence matrix, namely, the first column of the matrix with the two non-zero entries 1 and -1 and label this

column j. It will then remove j from the matrix to give us the deleted incidence matrix for the graph $\Sigma - e$. Once the program identifies j it will search for the rows with the non-zero entries and sum these two rows to combine them into one, thus, yielding the incidence matrix of the edge-contracted graph $\Sigma\backslash e$.

Example 2.



Consider a cycle of length three with two positive edges and one negative with the incidence matrix $\begin{bmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix}$. *Since the first column represents the first positive edge in our graph, then our matrix is reduced to* $\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & -1 \end{bmatrix}$ *by deletion and* $\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$ *by contraction as shown above.*

It is important to note that deletion-contraction cannot be applied to negative edges because when we delete a negative edge ij from Σ we count all possible colorings, including those where $x_i = -x_j$. When we contract a negative edge ij we only count the colorings where $x_i = x_j$. Thus, when we subtract the colorings of the contracted graph from the deleted one, the colorings where $x_i = -x_j$ are still included in $c_\Sigma(k)$, and recall that ij is negative, thus, the relation does not hold.

If a graph consists of no positive edges and no negative loops, then the graph only consist of negative edges. This is a problem because the deletion-contraction algorithm only works for positive edges and negative loops. Our program resolves this problem by implementing switching. As defined earlier, switching is the process of negating the signs of the edges originating from a particular vertex. Switching a graph that consists only of negative edges produces a new graph with at least one positive edge. As a result, this graph can now undergo the deletion-contraction process. A crucial implication of switching is that it preserves the chromatic polynomial of a graph as stated in Theorem 1. In other words, the chromatic polynomial of a graph before and after switching remains the same. Therefore, switching allows the program to delete and contract on a graph that originally contains only negative edges.

Once we compute the simpler chromatic polynomials, we sum these up to obtain the chromatic polynomial of our original graph.

3.3 Empty Graphs and Negative Loops

An *empty graph* is a graph with no edges and disconnected vertices. An empty graph is represented by an incidence matrix with one column and number of

rows equal to the number of vertices. All of the entries are zero because no edges exist between any of the vertices. The chromatic polynomial of an empty graph is $(2k+1)^n$ where n is the number of vertices. Given an incidence matrix, the program determines if the graph is empty by checking for number of columns and verifying that all entries are zero. If the graph is empty, the number of rows in the matrix is stored using the built-in SAGE function `nrows()`. The program returns the chromatic polynomial of the empty graph using the equivalence between number of rows and number of vertices, and the formula $(2k+1)^n$.
A graph containing a single negative loop is represented as a 1×1 incidence matrix with a 1 or -1 entry. The chromatic polynomial of a negative loop is $2k$ because the only restriction on coloring a vertex with a negative loop is that it cannot be zero since zero is the negative of itself. Given an incidence matrix, the program determines if the graph is a single negative loop by checking the column for a single non-zero entry. If the graph is a negative loop, the program returns $2k$.

3.4 Recursion

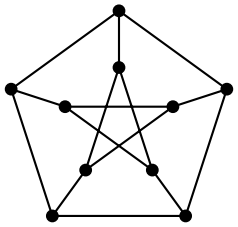
The deletion-contraction theorem defines the chromatic polynomial of Σ in terms of the chromatic polynomial of Σ with an edge e deleted and contracted. This makes the deletion-contraction relationship *recursive*, a recurence relation defines objects in terms of smaller objects of the same type. Recursive relations have a *base case* where the object is defined without recurring, the base case for the deletion-contraction relation is an empty graph because there are no edges to delete or contract. This base case ensures that the deletion-contraction algorithm does not delete and contract edges infinitely because the program terminates once the graph is empty and returns $(2k+1)^n$. The program also uses the negative loop as a base case, once the graph has been reduced to a negative loop it returns $2k$.
A *binary tree* is made up of vertices, where each vertex has a left and right pointer. These pointers can either point to an empty tree or another binary tree. The program creates a binary tree of incidence matrices, where the left pointer directs to an incidence matrix of the graph after deletion and the right pointer directs to an incidence matrix of the graph after contraction. The nodes in the binary tree that point to empty trees are called *leaves*. Leaves in our tree are base cases, either an empty graph or a negative loop. The program returns the sum of all the chromatic polynomials of the leaves of the tree and terminates.

4 The Final Product

The program can be run in any SAGE terminal. It takes an incidence matrix as an input and returns its chomatic polynomial as an expression. The code of the program is enclosed in Section 6, it can compute the chromatic polynomials of all six signed Petersen graphs in under fiteen minutes. Section 5 compiles the

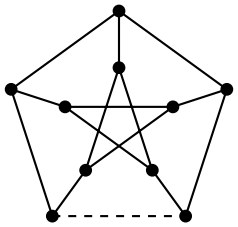
results along with the incidence matrices entered into the program.

5 Results: The Six Computed Chromatic Polynomials of The Petersen Graphs



$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1
\end{bmatrix}$$

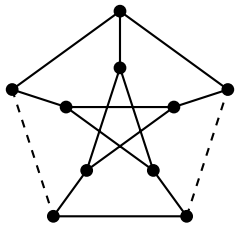
$$c_{P_1}(k) = 1024k^{10} - 2560k^9 + 3840k^8 - 4480k^7 + 3712k^6$$
$$-1792k^5 + 160k^4 + 480k^3 - 336k^2 + 72k$$



$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1
\end{bmatrix}$$

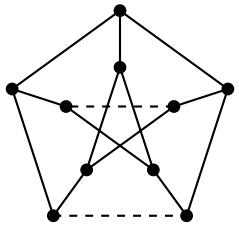
$$c_{P_2}(k) = 1024k^{10} - 2560k^9 + 3840k^8 - 4480k^7 + 3968k^6$$

$$-2560k^5 + 1184k^4 - 352k^3 + 48k^2$$



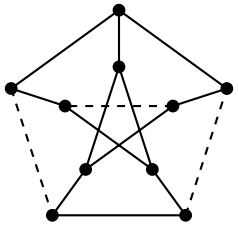
$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1
\end{bmatrix}$$

$$c_{P_3}(k) = 1024k^{10} - 2560k^9 + 3840k^8 - 4480k^7 + 4096k^6 \\ -2944k^5 + 1696k^4 - 760k^3 + 236k^2 - 40k$$



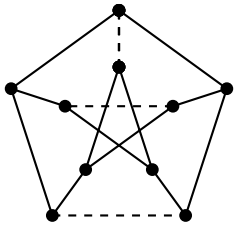
$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1
\end{bmatrix}$$

$$c_{P_4}(k) = 1024k^{10} - 2560k^9 + 3840k^8 - 4480k^7 + 4224k^6 \\ -3200k^5 + 1984k^4 - 952k^3 + 308k^2 - 52k$$



$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1
\end{bmatrix}$$

$$c_{P_5}(k) = 1024k^{10} - 2560k^9 + 3840k^8 - 4480k^7 + 4096k^6 \\ -3072k^5 + 1920k^4 - 960k^3 + 320k^2 - 48k$$



$$\begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1
\end{bmatrix}$$

$$c_{P_6}(k) = 1024k^{10} - 2560k^9 + 3840k^8 - 4480k^7 + 4480k^6 \\ -3712k^5 + 2560k^4 - 1320k^3 + 460k^2 - 90k$$

6 Code

This is the SAGE code used to run the program. This code can be loaded into any SAGE terminal and will compute the chromatic polynomial of any signed

graph. `chrom` is the main method which takes an incidence matrix and outputs the chromatic polynomial as an expression.

```
#All positive edges in columns have a 1 and -1 entry. Standardize
makes any positive edge column have the 1 come before the -1 when
reading the column top to bottom. All negative edges in columns have
two 1 entries or two -1 entires. Standardize makes any negative edge
column have two 1s. Negative loops have one 1 or -1 non-zero entry in
it's edge column. Standardize makes all negative loop columns have one
1. Standardizing our edges into one convention makes finding a
positive edge to delete-contract more efficient.
def standardize(G):
    G_rows=G.nrows()
    G_cols=G.ncols()
    for j in range(0,G_cols):
        for i in range(0,G_rows):
            if G[i,j] == 1:
                break
            if G[i,j] == -1:
                G.add_multiple_of_column(j,j,-2)
                break
    return G

#If there is a muti-edge in the peterson graph, meaning a column is
repeated, then delete the column and return a new incidence matrix.
def check(Gin):
    G=standardize(Gin) #
    G_cols=G.ncols()
    G_rows=G.nrows()
    l=G.columns()
    for i in range(0,G_cols-1):
        for j in range(i+1,G_cols):
            if G.column(G_cols-1-i)== G.column(G_cols-1-j):
                l.pop(G_cols-1-i)
                break
    B=matrix(l)
    C=B.transpose()
    return C



#Returns an ordered pair of the incidence matrix of the graph with an
edge deleted and the incidence matrix of the graph with an edge
contracted.
def DC(G):
    rows=G.nrows()
    cols=G.ncols()
    #If a single edge, return an empty graph with the same number of
vertices for the graph with the edge deleted. return an empty    graph
with one less vertex for the graph with the edge contracted.
    if cols == 1:
        C=matrix(QQ,rows,1,range(0))
        H=matrix(QQ,rows-1,1,range(0))
        return (C,H)
```

```
    #Else, find the first positive edge in the matrix when reading the
matrix from left to right. Delete and contract this edge
returning new incidence matrices.
    else:
        j=0
        #Increment through the columsn looking for a positive edge
        while j<cols:
            sum=0
            for i in range(0,rows):
                sum = G[i,j] + sum

            #If the non-sero entires in the column add to zero, this
is a positive edge. Delete it.
            if sum == 0:
                    delete_col=j
                    break
            j=j+1

      #If we found a positive edge then j will not be the number of
columns in the incidence matrix and we have deleted a column
      from the matrix. Create a new incidence matrix without this
column sotred in D, D represents a new graph with an edge deleted.
        if j != cols:
            l=G.columns()
            l.pop(delete_col)
            B=matrix(l)
            C=B.transpose()
            D=B.transpose()
            #Use this incidence matrix to contract an edge. This is
done by adding together the two rows which contained the deleted
column's non-zero entries creating a new incidence matrix H2.
            r1found = false
            for k in range(0,rows):
                if G[k,delete_col]!=0:
                    if r1found == false:
                        r_1=k
                        r1found = true
                    else:
                        r_2=k
          C.add_multiple_of_row(r_2,r_1,1)
            for p in range(0,cols-1):
                if abs(C[r_2,p])==2:
                    C[r_2,p]=1
            F=C.rows()
            F.pop(r_1)
            H=matrix(F)
            H1=check(H)
            D1=standardize(D)
            H2=standardize(H1)
            return (D1,H2)
```

```
        #If there was no positive edge in the matrix, then there
exists a negative loop. Delete Contract the negative loop.
        if j==cols:
            return c_neg_loop(G)

#Returns an ordered pair of the incidence matrix of the graph with a
negative loop deleted and the incidence matrix of the graph with a
negative loop contracted.
def c_neg_loop(G):
    G_rows=G.nrows()
    G_cols=G.ncols()
    E=G
    for j in range(0,G_cols):
        sum = 0
        for i in range(0,G_rows):
            sum=G[i,j] + sum
            if abs(G[i,j]) == 1:
                r1=i

        if abs(sum) == 1:
            l=G.columns()
            l.pop(j)
            B=matrix(l)
            D=B.transpose()
            m=D.rows()
            m.pop(r1)
            E=matrix(m)
            D1=standardize(D)
            E1=standardize(E)
            return (D1,E1)
            break
    if abs(sum)!= 1:
        return "All negative edges and no loops!"


#To ensure there is a positive edge or negative loop to delete-
contract, check to make sure one exists in the incidence matrix. If
not, we can switch a vertex by negating a row, creating a positive
edge or negative loop to delete and contract.
def switch(M):
    posEdge=False
    for item in M.columns():
        vNum=0
        sum=0
        nzExists=false
        for i in range (0,M.nrows()):
            if item[i] !=0:
                nzExists=True
                rowNum=i
            sum=sum+item[i]
        if nzExists==True and sum==0:
            posEdge=True
```

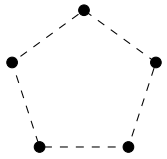
```
elif abs(sum)==1:
          posEdge=True
        elif nzExists==True and abs(sum)==2:
            vNum=rowNum
    if posEdge==False and M.ncols()!=0:
        M.add_multiple_of_row(vNum,vNum,-2)
    return M


#Returns the chromatic polynomial of a graph represented as an
incidence matrix.
def chrom(P):
    #Make sure the graph has a positive edge to delete and contract
and delete any multiple edges using switch and check.
    P=switch(P)
    P=check(P)
    #If the graph is a negative loop, return the chromatic polynomial.
    if len(P.columns())==1:
        if len(P.rows())==1:
           if P[0][0]==abs(1):
                return 2*x
    #If the graph is empty, return the chromatic polynomial.
    empty=true
    for item in P.columns()[0]:
     if item!=0:
           empty=false
    if empty==true:
     return ((2*x)+1)^(P.nrows())
    #Else, delete and contract the graph recursively calling chrom on
both the graph with the deleted edge and the graph with the
     contracted edge. Return the difference of the parts.
    (z,y)=DC(P)
    return expand(chrom(z))-expand(chrom(y))
```

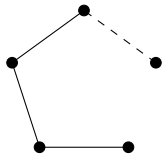
7 Appendix

Formulas for chromatic polynomials of cycles with all positive edges, cycles with one negative edge and paths were useful when verifying the deletion-contraction program was correct. These formulas are proved by induction on number of vertices n.

$\Sigma = (V, E, \sigma)$ is a *cycle* if and only if all $v \in V$ are incident to exaclty two vertices.



$\Sigma = (V, E, \sigma)$ is a *path* if and only if $v_1, v_2 \in V$ are incident to exactly one vertex and $i \in V/v_1, v_2$ are incident to exactly two vertices.



Claim: If Σ is a path then $c_\Sigma(k) = (2k+1)(2k)^{n-1}$.

Proof. Base Case: $n = 2$. $\Sigma = (V, E, \sigma) = (\{i, j\}, \{ij\}, \sigma)$.There are $(2k+1)$ choices for coloring vertex i, and $x_i \neq \sigma(ij)x_j$ so there are $(2k)$ color choices for j. Thus, $c_\Sigma(k) = (2k+1)(2k)^{n-1}$.
Inductive Hypothesis: Assume for Σ with $n-1$ vertices that $c_\Sigma(k) = (2k+1)(2k)^{n-2}$
Inductive Step:$\Sigma = (V, E, \sigma)$ has n vertices and $e \in E$ where e is incident with an end vertex.$\Sigma - e$ is a path along $n-1$ vertices with one disconnected vertex and Σ/e is a path with $n-1$ vertices. By the deletion-contraction theorem and inductive hypothesis:

$$\begin{aligned} c_\Sigma(k) &= c_{\Sigma-e}(k) - c_{\Sigma/e}(k) \\ &= (2k+1)(2k)^{n-2}(2k+1) - (2k+1)(2k)^{n-2} \\ &= (2k)^{n-2}(2k+1)[(2k+1)-1] \\ &= (2k+1)(2k)^{n-1} \end{aligned} \tag{1}$$

□

Claim: If $\Sigma = (V, E, \sigma)$ is a cycle graph with one negative edge e in E and all positive edges $E \backslash e$, then $c_\Sigma(k) = (2k)^n$.

Proof. : Base Case: $n = 1$. $\Sigma = (V, E, \sigma) = (\{i\}, \{ii\}, \sigma)$ has one negative loop at vertex i. $x_i \neq -x_i$, so $x_i \neq 0$. No other color is it's negative which leaves $2k$ colors to color i and $c_\Sigma(k) = 2k = (2k)^1 = (2k)^n$.
Inductive Hypothesis: Assume for Σ with $n-1$ nodes has $c_\Sigma(k)=(2k)^{n-1}$.
Inductive Step: Let Σ have n vertices. Use deletion-contraction theorem on any positive edge e in Σ. $\Sigma - e$ is a path with n vertices, so $c_{\Sigma-e} = (2k+1)(2k)^{n-1}$. Σ/e is a cycle with $n-1$ vertices and by inductive hypothesis $c_{\Sigma/e} = (2k)^{n-1}$.

$$\begin{aligned} c_\Sigma(k) &= c_{\Sigma-e}(k) - c_{\Sigma/e}(k) \\ &= (2k+1)(2k)^{n-1} - (2k)^{n-1} \\ &= (2k)^n + (2k)^{n-1} - (2k)^{n-1} \\ &= (2k)^n \end{aligned} \tag{2}$$

□

Claim: If $\Sigma = (V, E, \sigma)$ is a cycle graph with all positive edges in E, then $c_\Sigma(k)=(2k)^n + (-1)^n(2k)$.

Proof. : Base Case:n=2. $\Sigma = (\{i, j\}, \{ij, ji\}, \sigma)$. Any coloring where $x_i = x_j$ should not be counted in $c_\Sigma(k)$. We have $2k + 1$ choices for i and $2k$ choices for j because we must exclude x_i. So $c_\Sigma(k) = (2k + 1)(2k) = (2k)^2 + 2k = (2k)^2 + 1(2k) = (2k)^n + (-1)^n(2k)$
Inductive Hypothesis: Assume true for Σ with $n-1$ vertices.
Inductive Step:$\Sigma = (V, E, \sigma)$ and has n vertices and all positive edges. Contract and delete $e \in E$ so $\Sigma - e$ is a path with n vertices and Σ/e is a cycle with all positive edges and $n-1$ vertices.

$$\begin{aligned} c_\Sigma(k) &= c_{\Sigma-e}(k) - c_{\Sigma/e}(k) \\ &= (2k+1)(2k)^{n-1} - [(2k)^{n-1} + (-1)^{n-1}(2k)] \\ &= (2k)^n + (2k)^{n-1} - (2k)^{n-1} - [(-1)^{n-1}(2k)] \\ &= (2k)^n - [(-1)^{n-1}(2k)] \\ &= (2k)^n + (-1)^n(2k) \end{aligned}$$

□

8 Acknowledgements

We would like to thank Michael Young, Matthias Beck, and Ricardo Cortez for their guidance throughout the project. This work was supported by National Security Agency (NSA) grant H98230-11-1-0213 and National Science Foundation (NSF) grant DMS-1156499.

References

[1] W. A. Stein et al. *Sage Mathematics Software (Version 5.1)*. The Sage Development Team, 2012. `http://www.sagemath.org`.

[2] Thomas Zaslavsky. Matrices in the theory of signed simple graphs. In *Advances in discrete mathematics and applications: Mysore, 2008*, volume 13 of *Ramanujan Math. Soc. Lect. Notes Ser.*, pages 207–229. Ramanujan Math. Soc., Mysore, 2010.

[3] Thomas Zaslavsky. Six signed petersen graphs, and their automorphisms. *Discrete Mathematics*, 312(9):1558–1583, 2012.